Parallel Implementation of Bias Field Correction Fuzzy C-Means Algorithm for Image Segmentation

Noureddine AITALI
SSDIA Laboratory, ENSET-Mohammedia
Hassan II University
Casablanca, Morocco

Ahmed EL ABBASSI
PIM Laboratory, FST Errachidia
Moulay Ismail University, Morocco

Bouchaib CHERRADI
SSDIA Laboratory, ENSET-Mohammedia
Hassan II University, Casablanca, Morocco
Equipe STICE, CRMEF, El Jadida Morocco

Omar BOUATTANE
SSDIA Laboratory, ENSET-Mohammedia
Hassan II University
Casablanca, Morocco

Mohamed YOUSSFI
SSDIA Laboratory, ENSET-Mohammedia
Hassan II University
Casablanca, Morocco

***Abstract*—Image segmentation in the medical field is one of the most important phases to diseases diagnosis. The bias field estimation algorithm is the most interesting techniques to correct the in-homogeneity intensity artifact on the image. However, the use of such technique requires a powerful processing and quite expensive for big size as medical images. Hence the idea of parallelism becomes increasingly required. Several researchers have followed this path mainly in the bioinformatics field where they have suggested different algorithms implementations. In this paper, a novel Single Instruction Multiple Data (SIMD) architecture for bias field estimation and image segmentation algorithm is proposed. In order to accelerate compute-intensive portions of the sequential implementation, we have implemented this algorithm on three different graphics processing units (GPU) cards named GT740m, GTX760 and GTX580 respectively, using Compute Unified Device Architecture (CUDA) software programming tool. Numerical obtained results for the computation speed up, allowed us to conclude on the suitable GPU architecture for this kind of applications and closest ones.**

Keywords—Image segmentation; Bias field correction; GPU; Non homogeneity intensity; CUDA; Clustering

I. INTRODUCTION

In the medical image area, segmentation of anatomical structures is a key step for medical applications such as diagnostics, planning and act operation. Medical images contain a lot of information, and often few of structures are of interest. Segmentation allows visualization of the structures of interest and removing unnecessary information. It also enables structure analysis such as calculating the volume of a tumor, and performing feature-based image-to-patient as well as image-to-image registration, which is an important part of image guided surgery.

In magnetic resonance imaging, the in-homogeneities of intensities, called bias field, are caused by non-uniformities in the Radio Frequency RF field during the acquisition. The result is a shading effect where the pixel or voxel intensities of same tissue class vary slowly over the image domain. This shading can cause severe errors when attempting to segment corrupted images using intensity-based pixel classification methods. It has been shown that this shading is well modeled by the product of the original image and a smooth, very slowly varying multiplier field [1, 2].

In recent literature the authors in [3] proposed modified classical fuzzy c-mean algorithm to be able to get a handle on the intensity in-homogeneities and noisy image effectively. Authors in [4] have demonstrated that the combination of the iterative nonparametric non uniformity normalization and FCM correction method brightens the signal intensity of fatty tissues and that separates the histogram peaks between the fibro glandular and fatty tissues to permit an accurate segmentation between them. In the work of Huan Jun Ding and al [5] they have investigated the feasibility of volumetric breast density quantification with two computer assisted image segmentation methods on medical magnetic resonance imaging (MRI) scans of 40 postmortem breasts.

However, the use of their method is expensive in terms of time execution. In fact, most image segmentation methods proposed in the literature are computationally expensive, especially when run on large medical datasets, and requires powerful hardware to meet desired speed processing.

It also requires several techniques and algorithmic calculation models, which may be sequential or parallel using elementary processors, cellular automata, or neural networks.

In [6] Jaber Juntu and al have discussed two approaches to remedy the problem of the bias field corruption. The one can be used as a preprocessing step where the corrupted MRI image is re-established by dividing it by an estimated bias field signal using a surface fitting approach. The other approach explains how to edit the fuzzy c-means algorithm so that it can be used to segment an MRI image degraded by a bias field signal. The authors in [7] have proposed a fast spatially constrained kernel clustering algorithm in order to segment medical magnetic resonance imaging (MRI) brain images and correct the intensity in-homogeneities. VOVK Uroš and al [8] gave a review of different methods for correction of intensity in-homogeneity in MRI, they classified the methods according to the in-homogeneity correction strategy and different, qualitative and quantitative, evaluation approaches. Earlier, authors in [9] suggested a method that has been applied to 3T and 7T, they have achieved desirable results, and also their method was robust according to initialization. According to all this studies and developed algorithms for image processing, the reach of GPUs has made a revolution in the scientific community by using the power of parallel calculations that are well suited to this type of card.

Graphic processing units (GPUs) were originally created for rendering graphics. Recently, GPUs has emerged as co-processing units for Central Processing Units (CPU) and has become popular for general-purpose high performance computation (GP-GPU) which is mainly attractive and used by many researchers [10-12], to accelerate various digital signal processing applications, including medical image processing [13-15]. GPUs are composed of hundreds of processing cores, highly decoupled, able to achieve immense power of parallel computing. To take advantage of these multi-core architectures, these applications must be parallelized.

Among the images segmentation algorithms with intensity in-homogeneities correction on GPU architecture, authors in [16] proposed an extended mask-based version of the level set method with bias field, recently presented by Li et al. [17]. They develop CUDA implementations for the original full domain and the extended mask-based versions, and compare the methods in terms of speed, efficiency, and performance. The GPU implementation of their version allows a speed up of around 50−100 times for instance, for 512×512×128 slices.

Other researchers have suggested different implementation categories of FCM [18] on GPU in order to accelerate the running time. Anderson & al [19] suggested a GPU solution for the Fuzzy C-Means. They have used OpenGL and Cg to achieve approximately two orders of magnitude computational speed-up for some clustering profiles using an NVIDIA8800GPU card. Then they generalized the system for the use of non-Euclidean metrics [20]. On the other and they tried to provide computational intelligence researchers the skills necessary to exploit the low cost and high performance of GPUs with a minimum learning cost [21].

In other works authors implemented a parallel version of FCM on GPU using openGL and Cg language [22]. They reached about 2× speedup over the sequential implementation.

Rowinska and Goclawski [23] have focused on accelerating the clustering of the FCM algorithm on GPU using CUDA. They compared their implementation with its C++ sequential implementation, as well as a MATLAB version. In their papers FCM clustering was applied to segment polyurethane foam images. The NVIDIA GeForce GTX560 card was used to perform the parallel experiments, while Intel Core i3 processor was used for the sequential implementation. As a result, they showed that their parallel methods have achieved about 10×speed up over the sequential implementation and it was 50 to100× faster than the MATLAB version.

In [24] authors proposed a parallel implementation of brFCM which is a faster version of the standard FCM, they tested their algorithm on two GPU cards, Tesla M2070 and Tesla K20m, where the implementation provides about 2.24x speedup. The brFCM implementation achieves a speed up of 23.42x compared to the traditional FCM. Lately in [25] the authors have introduced a modified FCM algorithm that improved the calculations of the membership matrix and the centers update. Their upgrade version was implemented using CUDA on GPU hardware to rise: the execution time, the visual, and the segmentation efficiency. The experiments used different images of different sizes. They used GTX260 and Intel Core 2 Duo to run the GPU and CPU experiments. The authors achieved at least 10x improvements over the sequential FCM version.

Shalom et al. [26] proposed an implementation to improve the computational time of FCM on big data sets using GPU. The practical works were developed on a multidimensional yeast gene expression data set. The researchers stored the distance and membership matrices in the texture memory. The CPU carried out the initial step of the algorithm. Subsequently, the GPU held the most running time parts of the algorithm which are the iterative tasks such as distance calculations, membership calculations, and new cluster centers computations. They developed the experiments on two different GPU cards, GeForce 8500GT and GeForce 8800GTX. The comparison between the parallel and the serial implementations proves an up to 140x speedup on 8800GTX according to the CPU implementation. Moreover, the GPU implementation showed an up to 73x speedup on 8500GT.

In [27] the researchers implemented and analyzed a parallel dynamic functional connectivity (DFC). algorithm in GPU using two approaches, the first is thread-based and the second is block-based, moreover they have also parallelize the DFC using openMP on fMRI data, they have reached a speedup ranging from18.5**x** to 157x on GPU and 7.7**x** with openMP respectively.

In this paper, our contribution is mainly intended to parallelize the BCFCM algorithm [28] on massively parallel architecture. Our algorithm leads to a better accuracy on segmentation than FCM. But, it is more time consuming. In order to remedy this issue we have exploited the performance of NVIDIA graphic card (GPU) to implement a version that accelerates successfully the BCFCM. We will try later, to detail our method that gives more promising results.

The rest of this paper is organized as follows. In Section II, we summarized the fine-grained parallel model used (GPU of NVidia) and its software development environment (CUDA).

Section III presents a review of the sequential version of the clustering algorithm BCFCM proposed in [28]. Section IV, deals with our parallel implementation for the original and extended algorithms as well as the details of the parallelization. In Section V, we present our findings, compare the results and performance speed-ups. Section VI, concludes the paper and gives some perspectives for this work.

II. Parallel Architecture Model

In this section we will give a summary presentation of the computational model and the software development environment choosing for the implementation of the algorithm. It is related to GPU massively parallel architecture and it's SDK CUDA.

A. NVidia GPU Architecture

Modern GPUs are massively parallel processors that support a large number of elementary processors. They are particularly well suited for exploring the calculations on many data types that have high arithmetic intensity. Currently, GPU architecture is composed of an evolving set of processing units flows (SM, SMX Streaming Multiprocessor or: next-generation Streaming Multiprocessor). Such a multiprocessor contains a number of cores (scalar processor), a multithreaded instruction unit, number of registers, local memory and shared memory (Fig.1). The number of processor cores and multiprocessor depends on the architecture and model of the GPU.

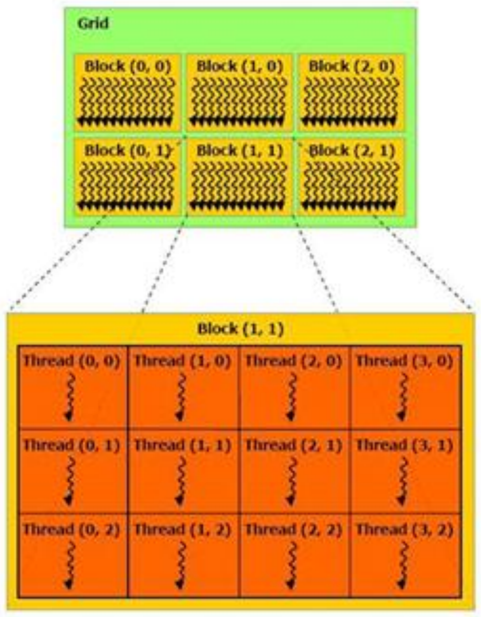

Fig. 1. NVIDIA GPU Grid, Block and Thread hierarchy in CUDA

The fundamental characteristic of the architecture of the GPU is that it has a massively parallel architecture that supports a large number of threads destined to end fine-grained calculation. Each parallelization scheme exploits the ability of mass GPU computing and provides a good load balancing, because each thread executes the same amount of computation.

B. CUDA: Compute Unified Device Achitecture

CUDA is a software development environment based on the C language for GPUs from NVIDIA unveiled in 2007 [29]. It's constituted by a parallel programming model and a set of dedicated instruction. The CUDA codes are compiled using the `NVCC` compiler [30]. It allows the programmer to define C functions, called kernels, which are executed in parallel by multiple CUDA threads (instantiation of a kernel). The programmer organizes these threads into a hierarchy of grids of thread blocks (Fig.1). A block of threads is a set of concurrent threads that can cooperate with each other through barrier synchronization and shared access. During execution, the threads can access data at different levels of hierarchy: registers, shared memory and global memory. The global memory is accessible by all threads, but its access time is about 500 times slower than the access time to the shared memory and registers.

The treatment of elementary processes (threads) on the GPU is not independent. Indeed, the threads are executed in groups called Warps, where in a warp given (32 threads), all threads execute the same instruction (SIMD).

III. Background: a Bias Field Correction (BC) Fuzzy C-Means (FCM) Algorithm (BCFCM)

The standard FCM [18] objective function for partitioning an MRI image containing $\{x_k\}_{k=1}^{N}$ pixels into C clusters is given by:

$$J = \sum_{i=1}^{C}\sum_{k=1}^{N} u_{ik}^{p} \left\| x_k - v_i \right\|^2 \qquad (1)$$

$u_{i,j}$: The degree of membership of data x_j in the cluster v_i,

v_i: The prototypes (or center) of the cluster i,

N: The total number of pixels in the MRI image

p: A weighting exponent parameter on each fuzzy membership value, it determines the amount of fuzziness of the resulting classification according to :

$$U\left\{ u_{ik} \in [0,1]\ , \sum_{i=1}^{C} u_{ik} = 1\ \forall k\ \ , 0 < \sum_{k=1}^{N} u_{ik} < N,\ \ \forall i \right\} \qquad (2)$$

The observed MRI signal is modeled as a product of the true signal generated by the underlying anatomy, and a spatially varying factor called the gain field.

$$Y_k = X_k G_k \qquad (3)$$

Where X_k and Y_k are the true and observed intensities at the k^{th} pixel, respectively, G_k is the gain field at the k^{th} voxel. The application of a logarithmic transformation to the intensities allows the artifact to be modeled as an additive bias field;

$$y_k = x_k + \beta_k \qquad (4)$$

Where x_k and y_k are the true and observed log-transformed intensities at the k^{th} pixel, respectively, and β_k is the bias field at the k^{th} pixel.

Ahmed et al [28] proposed a modification to (1) by introducing a term that allow the labeling of a pixel (Voxel) to be influenced by the labels in its immediate neighborhood. The modified objective function is given by:

$$J_m = \sum_{i=1}^{C}\sum_{k=1}^{N} u_{ik}^{p}\|y_k - \beta_k - v_i\|^2 + \frac{\alpha}{N_r}\sum_{i=1}^{C}\sum_{k=1}^{N} u_{ik}^{p}\left(\sum_{y_r \in N_k}\|y_r - \beta_r - v_i\|^2\right) \quad (5)$$

Where:

N_k: Set of neighbour's pixels that exist in a window around x_k.

N_r: Cardinal of N_k

α : Neighbours effect

The new membership function is then given by:

$$u_{ik} = \frac{1}{\sum_{j=1}^{C}\left(\frac{D_{ik} + \frac{\alpha}{N_r}\gamma_i}{D_{jk} + \frac{\alpha}{N_r}\gamma_j}\right)^{1/p-1}} \quad (6)$$

Where:

$$\gamma_i = \left(\sum_{y_r \in N_k}\|y_r - \beta_r - v_i\|^2\right) \quad (7)$$

And

$$D_{ik} = \|y_k - \beta_k - v_i\|^2 \quad (8)$$

The cluster prototype (centroid) updating is done by the expression:

$$V_i = \frac{\sum_{k=1}^{N} u_{ik}^{p}\left((y_k - \beta_k) + \frac{\alpha}{N_r}\sum_{y_r \in N_k}(y_r - \beta_r)\right)}{(1+\alpha)\sum_{k=1}^{N} u_{ik}^{p}} \quad (9)$$

The estimated bias field is given by the expression:

$$\beta_k = y_k - \frac{\sum_{i=1}^{C} u_{ik}^{p} v_i}{\sum_{i=1}^{C} u_{ik}^{p}} \quad (10)$$

Algorithm **1** explains the main steps of this algorithm.

Algorithm 1: Bias field Correction fuzzy C-Means Algorithm (B**C**FCM)

```
1: Set the parameters $C, p, N_r$ and $\alpha$ .
2. Choose the stop criteria: Error,
3: Initialize the centroïds vector V and estimated bias field $\beta$ .
4: repeat
5: Update the membership value U using Eq. (6)
6: Update the cluster center vector V using Eq. (9)
7: Update the bias field estimated matrix $\beta$ using Eq. (10)
8: until $\|V_{new} - V_{old}\| < Error$ .
```

IV. Parallel Fuzzy C-Means Algorithm for Bias Field Estimation and Segmentation (PBCFCM)

The main objective of the algorithm is to obtain simultaneously a bias field correction and image segmentation. However the expensive computation of this algorithm in its sequential version leads us to exploit the (SIMD) GPU architecture which is the adequate model for this kind of algorithms.Fig.2, illustrates the strategy used to distribute the data, at coarse-grained (Blocks) and fine-grained (threads) levels.

The main idea is to split the image over the GPU so that each pixel presented by one thread can execute its own instructions independently.

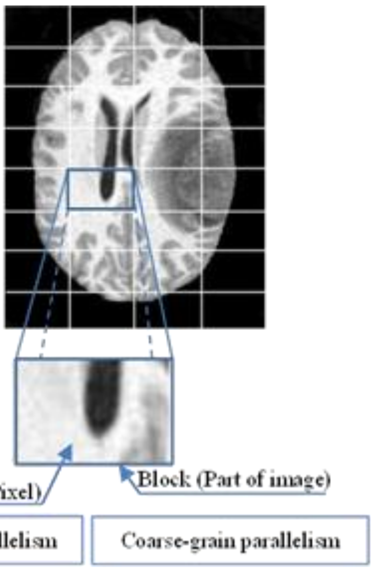

Fig. 2. Data distribution strategy for PBCFCM on GPU

For this algorithm we start with initializing the centroids vector and the variables, then allocate and transfer data from CPU to GPU before the loop iteration. Note that we have used two main kernels, one to compute the membership function and the second to compute the estimated bias field.

Once in the loop, we called the first kernel that computes both operations, the membership function and the expressions for updating centroids. Next step we update the cluster centers vector in CPU, then we transfer the new computed vector to the GPU in order to compute the new estimated bias field, and finally verify the criteria termination based on cluster variation.

The major problem of execution time in our case, is the large data transfer time that eat up the memory. While for small image size, data transfer between CPU and GPU is negligible and can even sometimes be preferable.

In this paper, we present a bias field correction fuzzy c-means implementation which exploits the shared memory for data and constant memory for centroids in addition of registers. While, the part assigned to be executed in CPU is the division operation for the results updating of cluster and the criteria termination test. Those two operations are assigned to CPU since they do not require processing power. The data transfer cost of cluster centers to constant memory may nearly be ignored compared to the iteration cost. Note that the allocation on device and data transfer are done only one time before starting the loop iteration, to avoid the latency caused by transferring data back and forth from the CPU and GPU.

The Parallel bias field Fuzzy c-means classification algorithm is achieved using the following stages:

Algorithm 2: Parallel bias field correction Fuzzy C-Means algorithm (PBCFCM).

Stage 1:
- Initialize the class centers *Vi*, (i = 1,...,C). This is carried out by selecting C points in the gray level scale [0,…,255].
- Read the input data file.
- Initialize the main variables of the algorithm in the CPU.

Stage 2: Copy the pixels data, the initial clusters and the estimated bias field to the GPU.
Stage 3: Allocate the memory of the membership matrix on the GPU.
Stage 4: Compute the membership values based on cluster center for each data member.
Stage 5: Compute expression needed for updating clusters on GPU.
Stage 6: Update the clusters centers on the CPU.
Stage 7: Transfer the new cluster centers to the GPU.
Stage 8: Compute the new estimated bias field according to the new clusters on GPU.
Stage 9: Check the difference between the current clusters value and its previous one. Test (If the stopping condition is reached) then exit; else return to *Stage 4*.
Stage 10: Output the final results.

V. Results and Discussion

A. Experiment setup

Algorithms on host (serial and parallel) were implemented using Microsoft VC++ program and CUDA 7.5 libraries. Sequential BCFCM algorithm is computed to obtain reference runtimes in C and compiled within Microsoft Visual Studio 2013 (Debug mode). Speed-up results were carried out on many devices: Intel(R) Core(TM) i7-4770 8 cores, 3.5GHz (CPU1), Intel(R) Core(TM) i7-4770 8 cores, 2.4 GHz (CPU2), GeForce GT740m, GTX760 and GTX580 GPUs. An example of specifications for tested processor and GPU equipment are summarized in Table 1 and the operating system was 64-bits Windows 7.

B. Implementation and Validation

To highlight the effectiveness of the proposed method, we extensively experiment both versions, sequential and parallel, of the clustering algorithm on different clinical brain images from online training database BRATS2012 [31] that offer a set of pathological cerebral MRI data. Before this implementation and in order to measure the performance of the proposed implementation in term of time execution, we have used the well-known *Lena* sample image and segment it into its optimal number of clusters which equals to 7 [32] on CPU and on GPU for different sizes that varies from 1024 to 6553600 pixels. This first assessment stage is done to identify perfectly the behavior of our implantation program on different computational GPU cards. In this section, the parameters set to validate and test the performances of our implementation for Lena image are as follow:

C=7, p=*2*, *Nr=8* and α =0.85 (α represents the neighbors effect as mentioned in [28] for low-SNR images).

To evaluate our implementation, we compared its performance to a sequential equivalent. For this purpose, runtime for serial execution was measured on the host processor that was obtained from single-core execution and referenced (TABLE.I). The GPU based computing duration for the same experiment parameters is compared with single-core timing. In GPU performance evaluation, the allocation on device and data transfer is done only one time before the loop iteration to avoid the latency caused by transferring data back and forth from the CPU and GPU. We calculate the time of the application after the file I/O, in order to show the speedup effect more clearly. The speedup results are normalized to the baseline which is the serial implementation.

TABLE I. Hardware Specification of a CPU and One of the Three GPUs (Core I7, GT 740m) Used in Experiments

Device	***Feature***	***Value***
CPU1	Name	i7 4770M
	Frequency	3.5 GHz
	Number of cores	8
	Installed memory	16 GO
GPU1	Name	GT 740
	Multiprocessor count	2
	CUDA cores	384
	Memory bus width	64 bits
	Warp Size	32
	Total global memory	2048MB
	Shared Memory per block	49152 octets
	Max threads per block	1024
	Max threads per multiprocessor	1024
	Memory clock rate	900Mhz

Speed up of the proposed parallel algorithm (PBCFCM) is related to the image size. The larger the size is, the more we get a better speed-up. This rule is not obvious because sometimes a low acceleration can be obtained. Furthermore and according to Fig.3, GPU cards do not have a linear acceleration, we can see that there is almost saturation, but there is always a slight speedup increase as shown in the following Fig.3.

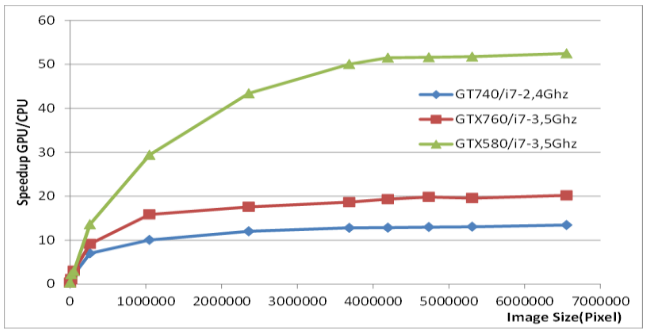

Fig. 3. Evolution of speed-up versus image size on different GPU devices for 32 bit configuration

Experimental results shown above illustrate two behaviors for the three types of GPU cards: GT 740M, GTX 760 and GTX 580, respectively.

Indeed:

- From 1024 pixels to 50526 pixels, the behavior is linear for the three devices,

- From 50526 up to 2359296 pixels the behavior is almost parabolic for GT 740m and GTX 760 while for the GTX 580, the parabolic behavior ends only at the value 3686400 pixels.

Beyond these experimental critical values, we note that, the speed-up tends slowly to the saturation state.

Also note that the maximum speed-up is achieved for the GTX 580 that is almost ***5x*** faster than GT 740M and ***2.5x*** faster than GTX 760.

The speed-ups reached for this figure are 52x, 20x and 13x for GTX 580, GTX 760 and GT 740 respectively for image size up to about to 7 Mega pixels.

According to our study, it seems clear that the performance speed-up depends on the strength of the GPU and the CPU configuration. Fig. 4 confirms the influence of the x64 configuration. Indeed treatment on images to this resolution is better than on x32 configuration (Fig. 3) for the same parallel implementation of the algorithm and different GPU.

Performance improvement is manifested on the three curves of the speed-up to an x64 configuration. The maximum values of speed-up obtained are of the order of 63x, 41x and 19x for GTX 580, GTX 760 and GT 740 respectively for image size up to about to 7 Mega pixels.

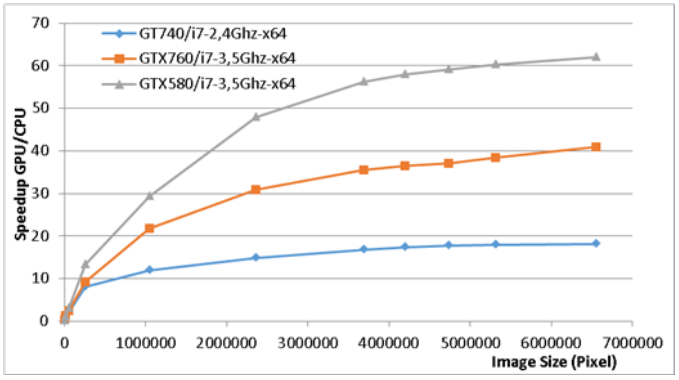

Fig. 4. Evolution of speed-up versus image size on different GPU devices for 64 bit configuration

In most cases and for large image sizes the suitable card having more performance in CUDA language is GTX 580 proved in this study.

To illustrate the impact of the resolution on the same graph, Fig. 5 shows the implementation speed up results of the parallel algorithm PBCFCM over sequential algorithm BCFCM, executed on three GPU cards and confronted the two types of configurations, namely win32 and x64.

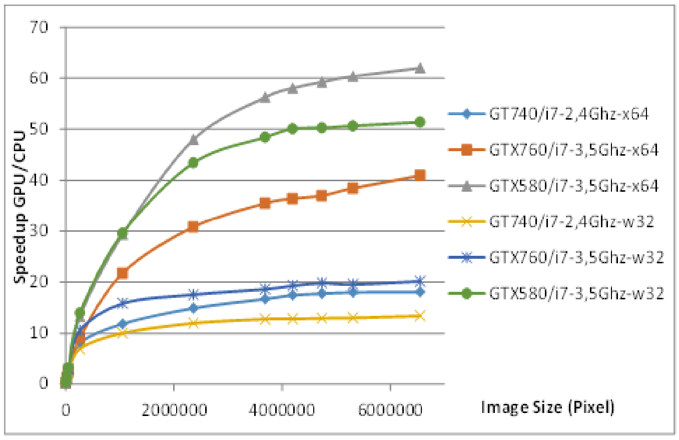

Fig. 5. Speed-up comparison for win32 and win x64 behavior on three GPU devices

Indeed, on the graph of Fig.6, the impact of a very high resolution is mainly shown on the curves where the resolution is x64 for the three types of GPU cards. On the other hand, for the same types of cards and even implementation of algorithm relating to x32 resolution, the speed-up is significantly lower than for x64 resolution. This feature is noticed on the three curves of the graph of figure 6 with x32 resolution. In order to evaluate the performance of the proposed GPU-BCFCM, we present in Fig.6 the variation of the execution time for iteration versus the picture size for the three types of the tested devices.

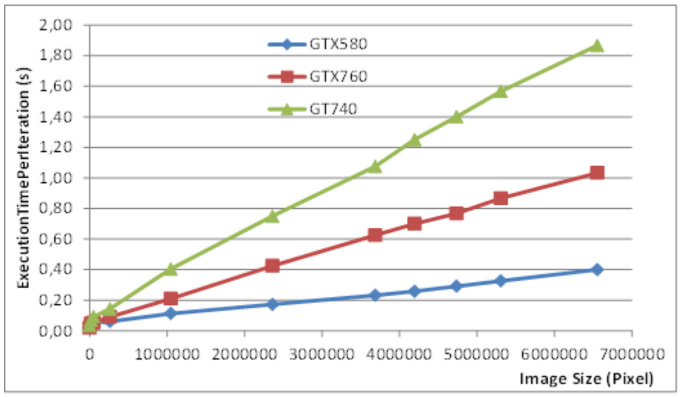

Fig. 6. The variation of execution time per iteration versus image size on different GPU devices

We can clearly see that the variation of execution time per iteration is straight line and strongly related on the performance of the used card. According to our measurements, significant slopes were notified in a decreasing order for: GT740M, GTX 760 and GTX 580 respectively. We found:

$$S_{GT740M} = 2.8\ 10^{-7}$$
$$S_{GTX760} = 1.5\ 10^{-7}$$
$$S_{GTX580} = 6.1\ 10^{-8}$$

All these slops are defined the ratio:

$$SGTxx = \frac{\text{Execution} - \text{Time} - \text{Per} - \text{Iteration}}{\text{image size on pixels}}$$

We note that the performance test of these cards corresponds to the lower slope. This is given for GTX 580 (S_{GTX580}) which has the fastest execution time.

We conclude that GTX 580 still better choose due to its performances compared to GTX 760 or GT 740M related to the slopes of experimental straight lines we introduced in this meaningful study.

The interest of the study that we conducted is clearly manifested for large image sizes. This is validated experimentally for different tests and different GPU cards. This study shows another aspect on performance analysis in terms of data storage effect on the performance of each card (data exchange inside memory card).

Another aspect of GPU performance can be sorted out by inspecting their computation power by testing them by a large amount of computation for a fixed data size. To do so, we extend the measurement for a fixed image size of 1024x1024, and run the FCM algorithm for a number of clusters ranging from 2 to 14 as in Fig.7 that shows the variation of speed-up according to number of clusters for different GPU devices.

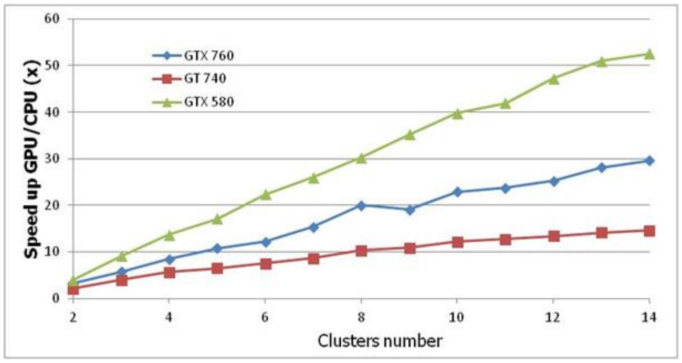

Fig. 7. Speed-up variation versus clusters number for *Lena* image size of 1024x1024

The results investigated in this paper are very relevant as long as they validate the highest importance of GPU processing optimization. This means that how to find optimal material for a given image processing requiring a given amount of memory and powerful ability of computation.

The speed varies almost linearly according to the number of cluster, for GT 740M, GTX 760 and GTX 580, respectively.

In fact, the values minima and maxima are ranging between 2.07x up to 14.64x for GT 740M, and between 3.31x up to 29.63x and finally for GTX 580 the speedup is ranging from 4.01x up to 52.6x.

The curves experimentally defined are close straight lines with dimensionless speed-up are defined per percentage and given as speedup ratios performance (SRP):

$$SRP = \frac{\text{Max speedup for 14 classes}}{\text{Min speedup for 2 classes}} * 100$$

So, for each GPU device we have:

SRP_{GT740M}= $SC14_{GT740M}/SC2_{GT740M}$ *100 =70.7%,

$SRP_{GTX760=}SC14_{GTX760}/SC2_{GTX760}$ *100 =89.5%,

SRP_{GTX580}=SC14GT740m/$SC2_{GT740m}$ *100 =131.1%.

According to these numerical results and parameters performance defined, we conclude that, GTX 580 is faster and powerful than GTX 760 which is also faster than GT740M.

Referring to Fig.8 and Fig.9, the image used for this measurement is 6.5536 million pixels of size. The figure below represents time execution in second versus the used devices categories.

In fact, for serial execution we got as result "62.6 s" which is the highest value comparing to GTX 580 of value 1.2s, GTX760 of value 3.1s and GT740M of value 5.6s respectively (Fig.8).

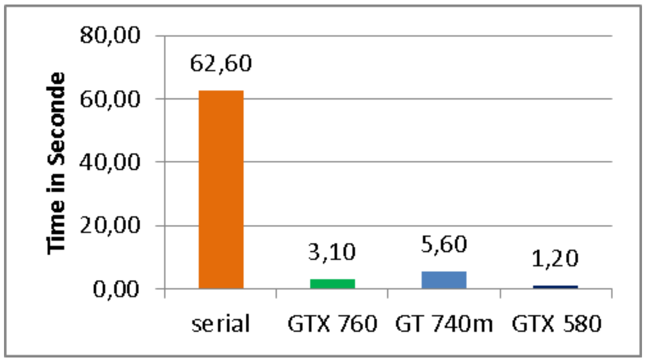

Fig. 8. Comparison of execution time for serial and parallel implementation on three devices

However, after the results expressed experimentally, it is clear that the parallel implementation is very benefic for images ranging from medium-sized to large. On the other hand, Fig.9 shows the speed-up reached in our experimental measurements, for GTX580, GTX760 and GT740M, the values obtained are 52.16x, 20.16x and 11.19x, respectively.

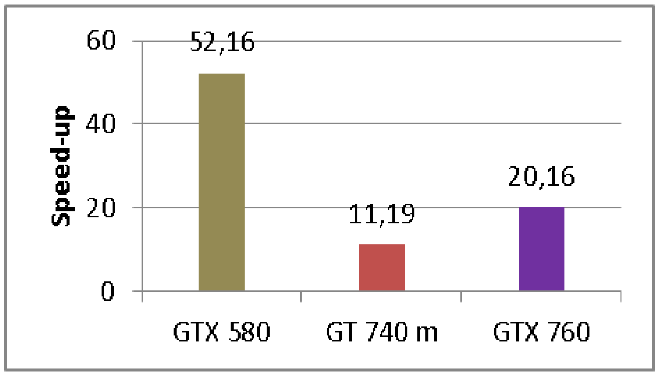

Fig. 9. Speedups of the parallel implementations over the serial one for L*ena* image size of.6.5536 million pixels

C. *Application example*

To illustrate the effectiveness of the proposed PBCFCM algorithm we present in fig.10 an application example on T2-weighted MRI image which contains melanoma tumor. This image was segmented into 5 clusters (Background, LCR, GM, WM and Tumor).

Figure 8 (b) shows the estimated Bias field, this was obtained by scaling the bias-field values from one to 255. Figure 8 (c) represents the corrected image after removing the bias field of figure 8(b). Figure 8 (d) shows the segmented image.

Both the parallel and the sequential algorithms was also applied to different MRI datasets image corrupted by bias field artefact and demonstrated its high performance in terms of correction and speed up according to the sequential version.

BCFCM is an algorithm that takes more time to estimate, correct and segment MRI images compared to fuzzy c-mean because it includes additional procedures that make segmentation and correction of intensity in-homogeneity based on neighborhoods. Its variants have high computational complexities that limit their applicability. The large image sizes processing needed by the medical applications motivates the researchers to increase the computation of the enhanced algorithms and take the advantages of the modern GPU of high processing ability.

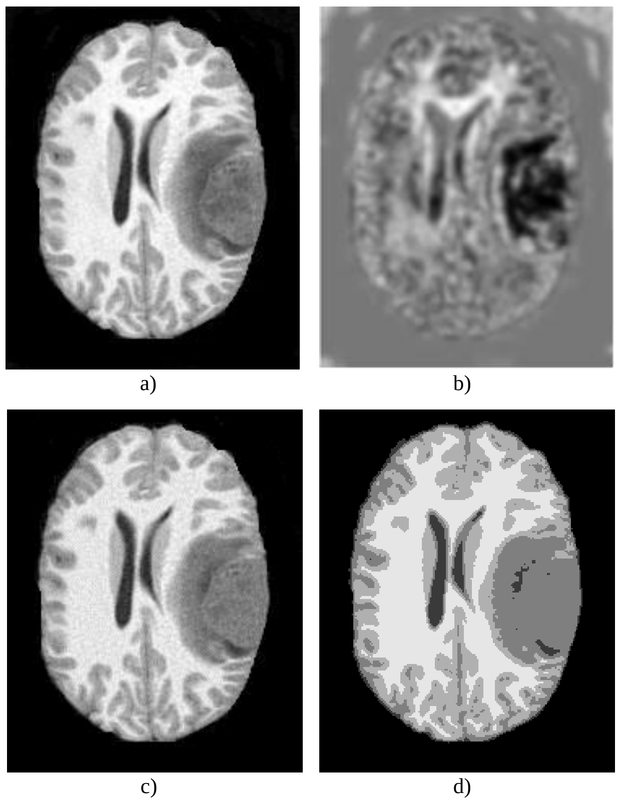

Fig. 10. Example of application. a) Test image, b) Bias-field estimations using PBCFCM algorithm: this was obtained by scaling the bias-field values from one to 255, c) Corrected image, d) 5 clusters segmented image

VI. Conclusion and Perspectives

In this paper we presented a parallel implementation of bias field algorithm PBCFCM which is an enhanced version of fuzzy C-means that correct the in-homogeneity intensity and segment the image simultaneously. In fact, many parameters and performance indices are introduced according to the realized measurements on different devices. Our results are translated into meaningful graphs, showing the impact of architecture on the card and the spot to which it was intended. The speed-up depends on the strength of the GPU, on the image size and on the number of clusters. Larger is the processing amount, the speed-up is higher.

For GTX 580 the speed-up increases until 52x versus the image size and also versus the number of clusters that is limited in this study to C=14. The obtained curves which are straight lines show that the speed-up is proportional to the number of clusters. For GTX 760 the speed-up increase until 20x versus image size and 29x versus the number of clusters, while for GT740M it reached 11x versus image size and 14x versus the number of clusters.

We can conclude from this study that whatever the performance of the graphic cards, it is necessary to take into account the amount of data to be processed before elaborating any parallel approach. The material performance test procedure remains obvious to fit the physical parallel architecture to the computational model algorithm and data inquiry.

As perspective to this work, we will establish an analytical model for the variation of the speed up versus the data size at first. This model will be improved by introducing the number of clusters especially for MRI classification domain. This study will be specifically launched for the three GPU cards used in our experiments.

References

[1] Rajapakse, J.C., Giedd, J.N., Rapoport, J.L., 1997. Statistical approach to segmentation of single-channel cerebral MR images. IEEE Trans. on Med. Imag. 16, 176-186. doi:10.1109/42.563663

[2] Unser, M., 1995. Multigrid adaptive image processing. In: Proceedings of the IEEE Conference on Image Processing, Vol. I, pp. 49-52. doi:10.1109/ICIP.1995.529036

[3] KANNAN S R, RAMATHILAGAM S, et PANDIYARAJAN R. Modified bias field fuzzy C-means for effective segmentation of brain MRI. In: Transactions on computational science VIII. Springer Berlin Heidelberg, 2010. p. 127-145.doi: 10.1007/978-3-642-16236-7_9

[4] LIN, Muqing, CHAN, Siwa, CHEN, Jeon-Hor, et al. A new bias field correction method combining N3 and FCM for improved segmentation of breast density on MRIa). Medical physics, 2011, vol. 38, no 1, p. 5-14. http://dx.doi.org/10.1118/1.3519869

[5] DING, Huanjun, JOHNSON, Travis, LIN, Muqing, et al. Breast density quantification using magnetic resonance imaging (MRI) with bias field correction: A postmortem study. Medical physics, 2013, vol. 40, no 12, p. 122305. http://dx.doi.org/10.1118/1.4831967

[6] JUNTU, Jaber, SIJBERS, Jan, VAN DYCK, Dirk, *et al.* Bias field correction for mri images. In : *Computer Recognition Systems*. Springer Berlin Heidelberg, 2005. p. 543-551.doi:10.1007/3-540-32390-2_64

[7] LIAO, Liang, LIN, Tusheng, et LI, Bi. MRI brain image segmentation and bias field correction based on fast spatially constrained kernel clustering approach. *Pattern Recognition Letters*, 2008, vol. 29, no 10, p. 1580-1588. doi:10.1016/j.patrec.2008.03.012

[8] VOVK, Uroš, PERNUŠ, Franjo, et LIKAR, Boštjan. A review of methods for correction of intensity inhomogeneity in MRI. *Medical Imaging, IEEE Transactions on*, 2007, vol. 26, no 3, p. 405-421. doi:10.1109/TMI.2006.891486

[9] JI, Ze-Xuan, SUN, Quan-Sen, et XIA, De-Shen. A modified possibilistic fuzzy c-means clustering algorithm for bias field estimation and segmentation of brain MR image. *Computerized Medical Imaging and Graphics*, 2011, vol. 35, no 5, p. 383-397. doi:10.1016/j.compmedimag.2010.12.001

[10] MOHANTY, Saraju P. GPU-CPU multi-core for real-time signal processing. In: Consumer Electronics, 2009. ICCE'09. Digest of

Technical Papers International Conference on. IEEE, 2009. p. 1-2. doi:10.1109/ICCE.2009.5012160

[11] ZHANG, Kang et KANG, Jin U. Real-time 4D signal processing and visualization using graphics processing unit on a regular nonlinear-k Fourier-domain OCT system. Optics express, 2010, vol. 18, no 11, p. 11772-11784.doi: 10.1364/OE.18.011772

[12] Jararweh Y, JarrahM, Hariri S (2012) Exploiting gpus for compute-intensive medical applications. In:Multimedia computing and systems (ICMCS), 2012 international conference on, IEEE, pp 29–34.doi:10.1109/ICMCS.2012.6320262

[13] Eklund, A., Dufort, P., Forsberg, D., & LaConte, S. M. (2013). Medical image processing on the GPU–Past, present and future. *Medical image analysis*, *17*(8), 1073-1094.doi:10.1016/j.media.2013.05.008

[14] Pratx, G., & Xing, L. (2011). GPU computing in medical physics: A review. *Medical physics*, *38*(5), 2685-2697. http://dx.doi.org/10.1118/1.3578605

[15] SMISTAD, Erik, FALCH, Thomas L., BOZORGI, Mohammadmehdi, *et al*. Medical image segmentation on GPUs–A comprehensive review. *Medical image analysis*, 2015, vol. 20, no 1, p. 1-18. doi:10.1016/j.media.2014.10.012

[16] T. Ivanovska, R. Laqua, L. Wang, H. Volzke, and K. Hegenscheid. “Fast Implementations of the Levelset Segmentation Method with Bias Field Correction in MR Images: Full Domain and Mask-Based Versions”. In: *Pattern Recognition and Image Analysis*. Springer Berlin Heidelberg, 2013. p. 674-681. doi:10.1007/978-3-642-38628-2_80

[17] Li, C., Huang, R., Ding, Z., et al.: A level set method for image segmentation in the presence of intensity inhomogeneities with application to MRI. IEEE Trans. on Image Processing 20, 2007–2016 (2011).doi:10.1109/TIP.2011.2146190

[18] Bezdek, J. C., Ehrlich, R., & Full, W. (1984). FCM: The fuzzy c-means clustering algorithm. Computers & Geosciences, 10(2), 191-203. doi:10.1016/0098-3004(84)90020-7

[19] Anderson, D., Luke, R., Keller, J. (2007). “Speedup of Fuzzy Clustering Through Stream Processing on Graphics Processing Units”, IEEE Trans. on Fuzzy Systems. doi:10.1109/TFUZZ.2008.924203

[20] Anderson, D., Luke, R. H., & Keller, J. M. (2007). Incorporation of non-euclidean distance metrics into fuzzy clustering on graphics processing units. In Analysis and Design of Intelligent Systems using Soft Computing Techniques (pp. 128-139). Springer Berlin Heidelberg. doi:10.1007/978-3-540-72432-2_14

[21] Anderson, (2008) Parallelisation of Fuzzy Inference on a Graphics Processor Unit Using the Compute Unified Device Architecture, The 2008 UK Workshop on Computational Intelligence, UKCI 2008, pp 1-6.

[22] Harris, C., & Haines, K. (2005, May). Iterative Solutions using Programmable Graphics Processing Units. In FUZZ-IEEE (pp. 12-18).

[23] Rowińska, Z., & Gocławski, J. (2012). Cuda based fuzzy c-means acceleration for the segmentation of images with fungus grown in foam matrices. Image Processing & Communications, 17(4), 191-200. doi: 10.2478/v10248-012-0046-7

[24] Al-Ayyoub, M., Abu-Dalo, A. M., Jararweh, Y., Jarrah, M., & Al Sa’d, M. (2015). A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation. The Journal of Supercomputing, 1-14. doi:10.1007/s11227-015-1431-y

[25] Li, H., Yang, Z., & He, H. (2014). An improved image segmentation algorithm based on GPU parallel computing. Journal of Software, 9(8), 1985-1990. doi:10.4304/jsw.9.8.1985-1990

[26] Shalom, S. A., Dash, M., & Tue, M. (2008, November). Graphics hardware based efficient and scalable fuzzy c-means clustering. In *Proceedings of the 7th Australasian Data Mining Conference-Volume 87* (pp. 179-186). Australian Computer Society, Inc.

[27] AKGÜN, Devrim, SAKOĞLU, Ünal, ESQUIVEL, Johnny, et al. GPU accelerated dynamic functional connectivity analysis for functional MRI data. Computerized Medical Imaging and Graphics, 2015, vol. 43, p. 53-63. doi:10.1016/j.compmedimag.2015.02.009

[28] M.N. Ahmed, N.A. Mohamed, A.A. Farag, T. Moriarty, A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data, IEEE Trans. Med. Imaging 21 (2002) 193–199.doi: 10.1109/42.996338

[29] http://www.nvidia.com/object/cuda_home_new.html.

[30] Nvcc: Cuda toolkit documentation. http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/.

[31] http://www2.imm.dtu.dk/projects/BRATS2012/.

[32] Ouadfel, S., Batouche, M., & Ahmed-Taleb, A. (2012). ACPSO: A Novel Swarm Automatic Clustering Algorithm Based Image Segmentation. In S. Ali, N. Abbadeni, & M. Batouche (Eds.) Multidisciplinary Computational Intelligence Techniques: Applications in Business, Engineering, and Medicine (pp. 226-238). Hershey, PA: Information Science Reference. 5. Chapter 14.